IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

| | |
|---|---|
| In re Appln. Of: | XU et al. |
| Serial No.: | 10/751,230 |
| Filed: | January 2, 2004 |
| For: | Method and System for More Effective Protein Three-Dimensional... |
| Docket: | GLH 08-896943 |

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

**SUBMISSION OF PRIORITY DOCUMENT**

Dear Sir:

Submitted herewith is the certified copy of Canadian Patent Application No. 2,415,584 in support of Applicants' priority claim under 35 USC 119.

Respectfully submitted,

Norman P. Soloway
Attorney for Applicants
Registration No. 24,315

**CERTIFICATE OF MAILING**

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on January 21, 2004 at Tucson, Arizona.

By: Kathryn Hume

HAYES SOLOWAY P.C.
130 W. CUSHING ST.
TUCSON, AZ 85701
TEL. 520.882.7623
FAX. 520.882.7643

175 CANAL STREET
MANCHESTER, NH 03101
TEL. 603.668.1400
FAX. 603.668.8567

Offic d la propriété int ll ctuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Prop rty Offic
An Agency of Industry Canada



# Bureau canadien des brevets

## Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

# Canadian Patent Office

## Certification

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial No: **2,415,584**, on January 2, 2003, by **BIOINFORMATICS SOLUTIONS INC.,** assignee of Jindo Xu and Ming Li, for "Protein Threading by Linear Programming".

Sylvie Grégoire
Agent certificateur/Certifying Officer

January 6, 2004
Date

Canada



OPIC CIPO

# PROTEIN THREADING BY LINEAR PROGRAMMING

JINBO XU, MING LI

December 31, 2002

Protein three-dimensional structure prediction through threading approach has been extensively studied and various models and algorithms have been proposed. In order to further explore ways to improve accuracy and efficiency of the threading process, this invention proposes a new method: protein threading via linear programming. Based on the contact map model of protein 3D structure, we formulate the protein threading problem as a large scale integer programming problem, then relax to a linear programming problem, and finally solve the integer program by a branch-and-bound method. The final solution is optimal with respect to energy functions incorporating pairwise interaction and allowing variable gaps. Our formulation often allows the linear program give integral solutions. The algorithm has been implemented as software package RAPTOR - RApid Protein Threading predictOR. Experimental results show that RAPTOR significantly outperforms other programs for fold recognition. The RAPTOR webserver is at http://www.cs.uwaterloo.ca/~j3xu/RAPTOR_form.htm

## 1 Introduction

The Human Genome Project has led to the identification of over 30 thousand genes in the human genome, which might encode, by some estimation, 100,000 proteins as a result of alternative splicing. To fully understand the biological functions and functional mechanisms of these proteins, the knowledge of their 3-D structures is required. The ambitious *Structural Genomics Initiatives*, launched by NIH in 1999, intends to solve these protein structures within the next ten years, through the development and application of significantly improved experimental and computational technologies.

A protein structure is typically solved using *x-ray crystallography* or *nuclear magnetic resonance spectroscopy (NMR)*, which are costly and very

time-consuming (ranging from months to years per structure) and is quite difficult for high-throughput production. The overall strategy of the NIH Structural Genomics Initiatives is to solve protein structures using experimental techniques like x-ray crystallography or NMR only for a small fraction of all the proteins and to employ computational techniques to model the structures for the rest of the proteins. The basic premise used here is that though there could be millions of proteins in nature, the number of unique structural folds is probably 2-3 (or even more) orders of magnitude smaller. Hence by strategically selecting the proteins with unique structural folds for experimental solutions, we can put the vast majority of other proteins "within the modeling distance" of these proteins. Model-based structure prediction techniques could play a significant role in helping to achieve the goal of the Structural Genomics Initiatives. *Protein threading* represents one of the most promising such techniques.

Protein threading can be used for both structure prediction and protein fold recognition, i.e., detection of homologous proteins. Numerous computer algorithms have been proposed for protein structure prediction, based on the threading approach. Based on the energy function models and computational methods, they can be grouped into three classes:

(1)The energy function does not include the pairwise interaction preferences explicitly. For this kind of model, a simple dynamic programming is employed to optimize the energy function. GenTHREADER[1] is a typical example. The prediction speed is fast, but theoretically, the prediction accuracy is worse than those incorporating pairwise interactions.

(2)The energy function includes the pairwise interaction preferences. However, it has been proved that this problem is NP-hard when variable gaps and pairwise interactions are considered simultaneously [2]. Some kinds of approximation algorithms are used to optimize the energy function. These methods include double dynamic programming[3], frozen approximation[4], and Monte Carlo sampling algorithm[5]. Unfortunately, T. Akutsu have proved that this problem is MAX-SNP-hard[6], which means that it cannot be approximated to arbitrary accuracy in polynomial time.

(3)The energy function includes the pairwise interaction preferences and an exact algorithm is designed to optimize the energy function. Xu *et al.* have proposed a divide-and-conquer method[7] which runs fast on simple protein template (interaction) topology but could take a long time for proteins with dense residue-residue interactions. In addition, this approach does not treat the following two special features explicitly: (i) interaction significance could be different from residues to residues; and (ii) interaction potentials could be heavily correlated with other non-pairwise scores such

as mutation scores and fitness scores.

Our main focus in this paper is on the development of a globally optimal and practically efficient threading algorithm based on the alignment model incorporating the pairwise interaction preferences explicitly and allowing variable gaps by using the integer programming approach. Integer programming formulation can fully exploit the abovementioned two special features of the pairwise interaction preferences. It allows us to use the existing powerful linear programming packages together with some branch and bound algorithm to rapidly arrive at the optimal alignment. To our knowledge, this is the first time that integer programming is applied to protein threading.

## 2 Alignment Model

We represent the amino acid sequence, of length $m$, of a protein template by $t_1 t_2 \dots t_m$ and the query sequence, of length $n$, by $s_1 s_2 \dots s_n$. In formulating the protein threading problem, we follow a few basic assumptions widely adopted by the protein threading community [7]. We assume that:

(1) Each template consists of a linear series of cores with the connecting loops between the adjacent cores. Each core is a conserved segment of an $\alpha$-helix or $\beta$-sheet secondary structure among the protein's homologs. Although the secondary structure is often conserved, insertion or deletion may occur within a secondary structure. So we only keep the most conserved part. Let $c_i = core(head_i, tail_i)$ denote all cores of one template, where $i = 1, 2, \dots, M$ with $M$ being the number of the cores, and $1 \leq head_1 \leq tail_1 < head_2 \leq tail_2 < \dots < head_M \leq tail_M \leq m$. The region between $tail_i$ and $head_{i+1}$ is a loop. The length of $c_i$ is $len_i = tail_i - head_i + 1$. Let $loc_i$ denote the sum of the length of all cores before $c_i$, i.e., $loc_i = \sum_{j=1}^{i-1} len_j$.

(2) When aligning a query protein sequence with a template, alignment gaps are confined to loops, *i.e.*, the regions between cores or the two ends of the template. The biological justification is that cores are conserved so that the chance of insertion or deletion within them is very little.

(3) We consider only interactions between core residues. It is generally believed that interactions involving loop residues can be ignored as their contribution to fold recognition is relatively insignificant. We say that an interaction exists between two residues if the spatial distance between their $C_\alpha$ atoms is within $7A$ and they are at least 4 positions apart in the template sequence. *We say that an interaction exists between two cores if there exists at least one residue-residue interaction between the two cores.*

Our threading energy function consists of environment fitness score $E_s$, mutation score $E_m$, secondary structure compatibility score $E_{ss}$, gap penalty $E_g$ and pairwise interaction score $E_p$. The overall energy function $E$ has the following form:

$$E = W_m E_m + W_s E_s + W_p E_p + W_g E_g + W_{ss} E_{ss},$$

where $W_m, W_s, W_p, W_g, W_{ss}$ are weight factors to be determined by training.

Global alignment and global-local alignment methods are employed to align one template to one sequence. For the detailed description, please refer to Fischer's paper[8]. In the case that the template size is larger than the query sequence size, it is possible that some cores at the two ends of the template cannot be aligned to the sequence. But we can always extend the sequence by adding some "artificial" amino acids to the two ends of the sequence to make all cores are aligned to the (extended) sequence. All scores involving those extended positions are set to be 0.

## 3 Formulation

**Definition 3.1** *We use an undirected graph $CMG = (V, E)$ to denote the contact map graph of a protein template structure. Here, $V = \{c_1, c_2, ..., c_M\}$ where $c_i$ represents $i^{th}$ core, and $E = \{(c_i, c_j) | \text{there is an interaction between } c_i \text{ and } c_j, \text{or } |i-j| = 1\}$.*

For simplicity, when we say that core $c_i$ is aligned to position $s_j$, we always mean that core $c_i = (head_i, tail_i)$ is aligned to segment $(s_j, s_{j+len_i-1})$. In order to speed up the search, RAPTOR employs some knowledge-based filtering process proposed in PROSPECT[7] that indicates certain alignments as *invalid*.

**Definition 3.2** *Let $B$ denote the alignment bipartite graph of one threading pair. Each core of the template corresponds to one vertex in $B$, labeled as $c_i (i = 1, 2, \ldots, M)$, each residue in the query sequence corresponds to one vertex in $B$, labeled as $s_j (j = 1, 2, \ldots, n)$. The edges of $B$ consist of all valid alignments (after initial filtering) between each core and each sequence position. The edges of $B$ are also called the alignment edges.*

**Definition 3.3** *For any two different edges $e_1 = (c_{i_1}, s_{j_1})$ and $e_2 = (c_{i_2}, s_{j_2})$ in an alignment bipartite graph $B$, if $(loc_{i_1} - loc_{i_2}) \times (s_{j_1} + loc_{i_2} - loc_{i_1} - s_{j_2}) \leq 0$, then we say $e_1$ and $e_2$ are in* conflict.

The proof of the following three lemmas is omitted due to space limit.

**Lemma 3.1** *For any three different edges* $e_r = (c_{i_r}, s_{j_r})$, $r = 1,2,3$ *and* $loc_{i_1} < loc_{i_2} < loc_{i_3}$, *if* $e_1$ *conflicts with* $e_2$ *and* $e_2$ *conflicts with* $e_3$, *then* $e_1$ *conflicts with* $e_3$.

**Lemma 3.2** *For any three different edges* $e_r = (c_{i_r}, s_{j_r})$, $r = 1,2,3$ *and* $loc_{i_1} < loc_{i_2} < loc_{i_3}$, *if* $e_1$ *does not conflict with* $e_2$ *and* $e_2$ *does not conflict with* $e_3$, *then* $e_1$ *does not conflict with* $e_3$.

**Lemma 3.3** *For any three different edges* $e_r = (c_{i_r}, s_{j_r})$, $r = 1,2,3$ *and* $loc_{i_1} < loc_{i_2} < loc_{i_3}$, *if* $e_1$ *conflicts with* $e_3$, *then* $e_2$ *conflicts with* $e_3$ *or* $e_2$ *conflicts with* $e_1$.

**Definition 3.4** *An alignment is called a valid alignment if: (1) each core of the template is aligned to some position of the (extended) sequence*[1]*; (2)For any two different cores* $c_1$ *and* $c_2$, *their two alignment edges do not conflict in the alignment graph.*

Let $x_{i,l}$ be a boolean variable such that $x_{i,l} = 1$ if and only if core $c_i$ is aligned to position $s_l$. Similarly, for any $(c_{i_1}, c_{i_2}) \in E(CMG)$, let $y_{(i_1,l_1),(i_2,l_2)}$ indicate the pairwise interactions between $x_{i_1,l_1}$ and $x_{i_2,l_2}$ if the two edges $(c_{i_1}, s_{l_1}), (c_{i_2}, s_{l_2})$ do not conflict. $y_{(i_1,l_1),(i_2,l_2)} = 1$ if and only if $x_{i_1,l_1} = 1$ and $x_{i_2,l_2} = 1$. We say $y_{(i_1,l_1),(i_2,l_2)}$ is generated by $x_{i_1,l_1}$ and $x_{i_2,l_2}$.

The $x$ variables are called the alignment variables and $y$ variables are called the interaction variables. Let $D[i]$ denote all valid query sequence positions that $c_i$ could be aligned to. Let $R[i,j,l]$ denote all valid alignment positions of $c_j$ given $c_i$ is aligned to $s_l$.

Now the objective function of the protein threading problem can be formulated as follows:

$$\min W_m E_m + W_s E_s + W_p E_p + W_g E_g + W_{ss} E_{ss}, \tag{1}$$

$$E_m = \sum_{i=1}^{M} \sum_{l \in D[i]} [x_{i,l} \times \sum_{r=0}^{len_i - 1} Mutation(head_i + r, l + r)], \tag{2}$$

$$E_s = \sum_{i=1}^{M} \sum_{l \in D[i]} [x_{i,l} \times \sum_{r=0}^{len_i - 1} Fitness(head_i + r, j + r)], \tag{3}$$

[1] As mentioned before, global and global-local alignment are employed.

$$E_{ss} = \sum_{i=1}^{M} \sum_{l \in D[i]} [x_{i,l} \times \sum_{r=0}^{len_i - 1} SS(t_{head_i + r}, s_{j+r})], \quad (4)$$

$$E_p = \sum_{1 \le i < j \le M, (c_i, c_j) \in E(CMG)} \sum_{l \in D[i]} \sum_{k \in R[i,j,l]} y_{(i,l),(j,k)} P(i,j,l,k), \quad (5)$$

$$P(i,j,l,k) = \sum_{u=0}^{len_i - 1} \sum_{v=0}^{len_j - 1} \delta(t_{head_i + u}, t_{head_j + v}) Pair(l+u, k+v) \quad (6)$$

$$E_g = \sum_{i=1}^{M} \sum_{l \in D[i]} \sum_{k \in R[i,i+1,l]} y_{(i,l),(i+1,k)} G(i,l,k), \quad (7)$$

where $\delta(t_u, t_v) = 1$ if there is an interaction between residues at position $u$ and $v$ in the template, otherwise 0. $G(i,l,k)$ is the gap potential between $c_i$ and $c_{i+1}$ when they are aligned to query sequence position $l$ and $k$ respectively. $G(i,l,k)$ could be computed by dynamic programming in advance given $i, l, k$.

The constraint set is as follows:

$$\sum_{j \in D[i]} x_{i,j} = 1, i = 1, 2, \ldots, M; \quad (8)$$

$$\sum_{l \ge l_0, l \in D[i]} x_{i,l} + \sum_{k \in D[i+1] - R[i,i+1,l_0]} x_{i+1,k} \le 1, l_0 \in D[i], i = 1, 2, \ldots, M-1; \quad (9)$$

$$\sum_{k \in R[i,j,l]} y_{(i,l),(j,k)} \le x_{i,l}, \forall l \in D[i], i, j = 1, 2, \ldots, M; \quad (10)$$

$$\sum_{l \in R[j,i,k]} y_{(i,l),(j,k)} \le x_{j,k}, \forall k \in D[j], i, j = 1, 2, \ldots, M; \quad (11)$$

$$\sum_{k \in R[i,j,l]} y_{(i,l),(j,k)} \ge x_{i,l} + \sum_{k \in R[i,j,l]} x_{j,k} - 1, l \in D[i], i, j = 1, 2, \ldots, M; \quad (12)$$

$$\sum_{l \in R[j,i,k]} y_{(i,l),(j,k)} \ge x_{j,k} + \sum_{l \in R[j,i,k]} x_{i,l} - 1, k \in D[j], i, j = 1, 2, \ldots, M; \quad (13)$$

$$x_{i,j} \ge 0, j \in D[i], i = 1, 2, \ldots, M; \quad (14)$$

$$y_{(i,l)(j,k)} \ge 0, \forall l \in D[i], k \in D[j], i, j = 1, 2, \ldots, M. \quad (15)$$

Constraint 8 says that one core can be aligned to a unique sequence position. Constraint 9 forbids the conflicts between the adjacent two cores. Therefore, based on lemma 3.2, this constraint can guarantee that there are no conflicts between any two cores if variable $x$ and $y$ are integral. Constraints 10 and 11 say that at most one interaction variable can be 1 between any two cores that have interactions between each other. Constraints 12 and 13 enforce that if two cores have their alignments to the sequence respectively and also have interactions between them, then at least one interaction variable should be 1. Constraints 8,14 and 15 guarantee $x$ and $y$ to be between 0 and 1 when this problem is relaxed to linear program.

There is another set of more obvious constraints which can replace Constraints 9-13. They are:

$$x_{i,l} + x_{i+1,k} \leq 1, \forall k \in D[i+1] - R[i, i+1, l]; \tag{16}$$

$$y_{(i,l)(j,k)} \leq x_{i,l}, k \in R[i,j,l], (c_i, c_j) \in E(CMG); \tag{17}$$

$$y_{(i,l)(j,k)} \leq x_{j,k}, l \in R[j,i,k], (c_i, c_j) \in E(CMG); \tag{18}$$

$$y_{(i,l)(j,k)} \geq x_{i,l} + x_{j,k} - 1, (c_i, c_j) \in E(CMG); \tag{19}$$

Constraint 16 forbids the conflict between two neighboring cores and Constraints 17-19 guarantee that one interaction variable is 1 if and only if its two generating $x$ variables are 1. Constraints 16-19 can be inferred from Constrains 9-13. Conversely, it is not true. Therefore, Constraints 16-19 are weaker than Constraints 9-13.

In order to improve running time, we found yet another set of Constraints 20 and 21 from which both 9-13 and 16-19 can be inferred (proofs omitted due to space limitation).

$$\sum_{k \in R[i,j,l]} y_{(i,l)(j,k)} = x_{i,l}, (c_i, c_j) \in E(CMG); \tag{20}$$

$$\sum_{l \in R[j,i,k]} y_{(i,l)(j,k)} = x_{j,k}, (c_i, c_j) \in E(CMG); \tag{21}$$

Constraints 20 and 21 imply that one $x$ variable is 1 is equivalent to that one of the $y$ variables generated by it is 1 . These two are the strongest constraints. Experimental results show that our algorithm with Constraints 20 and 21 (combining with Constraints 8,14 and 15) runs significantly faster. Our program RAPTOR uses 20 and 21 by default.

# 4 RAPTOR – Implementation

## 4.1 Scoring System

We calculated the averaged energy over a set of homologous sequences, as demonstrated in PROSPECT-II[9]. Given a query sequence of length $n$, an $n \times 20$ frequency matrix $PSFM$ is calculated by using PSI-BLAST[10] with maximum iteration number being set to 5. Each column of this matrix describes the occurring frequency of 20 amino acids at this position. Assume a template position $i$ is aligned to the sequence position $j$. Then the mutation score and fitness score are calculated as follows.

$$Mutation(i,j) = \sum_a p_{j,a} M(t_i, a)$$

$$Fitness(i,j) = \sum_a p_{j,a} F(env_i, a)$$

where $p_{j,a}$ represents the occurring frequency of amino acid $a$ at sequence position $j$, $M(a,b)$ represents the mutation potential between two amino acid $a$ and $b$ which is taken from PAM250 matrix[11], $F(env, a)$ denote the fitness potential when amino acid $a$ is placed into environment $env$.

The 9 combinations of three secondary structure types ($\alpha$-helix, $\beta$-strand and coil) and three solvent accessibility levels are used to define the local environments of a position in the template. The boundaries between thre solvent accessibility levels are at 7% and 37%. Secondary structure and solvent accessibility assignments are all taken from FSSP database[12].

The gap penalty function is assumed to be an affine function, i.e., a gap open penalty plus a length-dependent gap extension penalty. Gap open penalty is set at 10.6 and gap elongation penalty is 0.8 per single gap[13]. We use PSIPRED [14] to predict the secondary structure of the query sequence.

If the two ends of an interaction are aligned to $j_1^{th}$ and $j_2^{th}$ positions of the query sequence respectively, then the pair score for this interaction is given by:

$$Pair(j_1, j_2) = \sum_a p_{j_1,a} \sum_b p_{j_2,b} P(a,b)$$

where $P(a,b)$ denotes the pairwise interaction potential between two amino acids $a$ and $b$. $F, P$ are taken from PROSPECT-II[9].

## 4.2 Branch-and-Bound Method

We use a branch-and-bound algorithm to solve the above integer programming problem. First we relax the above integer program by allowing all $x$

and $y$ to be real between 0 and 1 and solve the resulting linear program. If the solution $(x^*, y^*)$ of the linear program is integral, then we get the optimal solution. Otherwise, we select one non-integral variable according to some criterion, and generate two subproblems by setting it to 0 and 1 respectively. These two subproblems are solved recursively. More details on solving integer programming problem can be found in [15]. IBM OSL(Optimization and Solution Library) package is used to implement this process.

### 4.3 Weight Training

The weight factors $W_m, W_s, W_{ss}, W_g, W_p$ are chosen through optimizing the overall alignment accuracy. The optimal alignment accuracy does not necessarily imply the best fold recognition capability though. In the following subsection, an SVM (Support Vector Machine) method is used to carry out fold recognition. A set of 95 structurally-aligned protein pairs are chosen from Holm's test set[16] as the training samples, each of which only has fold-level similarity. The alignments generated by RAPTOR is compared with the structural alignments generated by SARF[17]. An alignment for a residue is regarded as correct if it is within 4 residue shift away from the correct structure-structure alignment by SARF. The overall alignment accuracy is defined as the ratio between the number of the correctly-aligned positions of all threading pairs and the number of the maximum alignable positions. Our objective is to maximize the overall alignment accuracy. A genetic algorithm plus a local pattern search method implemented in DAKOTA[18] is used to search for the optimal weight factors. We attained 56% alignment accuracy over this set of training pairs. A set of 1100 protein pairs which are in the fold-level similarity is also generated from Holm's test set[16] to test the weight factors and 50% alignment accuracy is attained. We have also selected 95 structurally-aligned protein pairs from Holm's test set, each of which is in superfamily-level or family-level similarity, 80% alignment accuracy is achieved when the same set of weight factors is used.

### 4.4 $z$-score and Fold Recognition

After threading one pair of sequence and template, $z$-score is calculated according to the method proposed in paper[19] to cancel out the composition bias. Let $z_{raw}$ denote this kind of $z$-score. However, since the accurate $z_{raw}$ is expensive to compute, we just approximate it by (i) fixing the alignment positions; (ii) shuffling the query sequence randomly; and (iii) calculating the alignment scores based on the existing alignment rather than doing optimal

alignments again and again. The free software SVM light[20] with RBF kernel is employed to adjust the approximate $z$-score. Due to space limit, we refer the reader o Vapnik's book[21] for a comprehensive tutorial of SVM. A set of 60000 training pairs formed by all-against-all threading between 300 templates (randomly chosen from the FSSP database) and 200 sequences (randomly chosen from Holm's test set [16]) is used as the training samples of our SVM model. The relationship between two proteins is judged based on SCOP database[22]. If one pair is in at least fold-level similarity, then it is treated as a positive example, otherwise a negative example. Each of training samples consists of the following features: (1)$z_{raw}$; (2) the sequence size; (3) the template size; (4) the number of cores in the template; (5) the sum of the core size in the template; (6) the number of aligned cores; (7) the number of aligned positions; (8) the number of identical residues; (9) the number of contacts with both ends on the aligned cores; (10) the number of cut contacts with one end on the aligned cores and the other on the unaligned cores; (11) the total score; (12) mutation score; (13) singleton fitness score; (14) gap score; (15) secondary score; (16) pair score. Given one threading result, SVM outputs a real value. The value greater than 0 means this threading pair is in at least fold-level similarity. We do not use this directly due the abundance of the false negatives. We calculate the final $z$-score for each query sequence. For all threading pairs of one given sequence, let $o_1, o_2, ..., o_q$ denote the outputs from SVM model. The final $z$-score is calculate by $\frac{o_i - u(o)}{std(o)}$, where $u(o)$ is the mean value of $o_i$ and $std(o)$ is the standard deviation of $o_i$. Daniel Fischer's benchmark[8] is used to fix the parameters of the model.

## 5 Preliminary Experimental Results

Fischer's benchmark consists of 68 target sequences and 301 templates. RAPTOR ranks 56 pairs out of 68 pairs as top 1, achieving 82% prediction rate, while the previous best was 76.5%.

The fold recognition performance of RAPTOR was further tested on Lindahl's benchmark set consisting of 976 protein sequences [23]. By threading them all against all, there are totally $976 \times 975$ pairs. We measure RAPTOR's performance in three different similarity levels: fold, superfamily and family. The results are shown in Table 1. The results of other methods are taken from Shi et al's paper[24].

As shown in Table 1, the performance of RAPTOR at the fold level is much better than the others. At the superfamily level, RAPTOR per-

| method | Family | | Superfamily | | Fold | |
|---|---|---|---|---|---|---|
| | Top 1 | Top 5 | Top 1 | Top 5 | Top 1 | Top 5 |
| **RAPTOR** | **75.2** | **77.8** | **39.3** | **50.0** | **25.4** | **45.1** |
| **RAPTOR-np** | **68.9** | **72.8** | **34.0** | **49.7** | **19.0** | **36.6** |
| FUGUE | 82.2 | 85.8 | 41.9 | 53.2 | 12.5 | 26.8 |
| PSI-BLAST | 71.2 | 72.3 | 27.4 | 27.9 | 4.0 | 4.7 |
| HMMER-PSIBLAST | 67.7 | 73.5 | 20.7 | 31.3 | 4.4 | 14.6 |
| SAMT98-PSIBLAST | 70.1 | 75.4 | 28.3 | 38.9 | 3.4 | 18.7 |
| BLASTLINK | 74.6 | 78.9 | 29.3 | 40.6 | 6.9 | 16.5 |
| SSEARCH | 68.6 | 75.7 | 20.7 | 32.5 | 5.6 | 15.6 |
| THREADER | 49.2 | 58.9 | 10.8 | 24.7 | 14.6 | 37.7 |

Table 1: The performance of RAPTOR at three different similarity levels

forms a little bit worse than FUGUE [24], the best method (for superfamily and family level) listed in this table. However, at the family level, RAPTOR performs better than only THREADER, which means that RAPTOR is superior in recognizing fold-level similarity but bad in doing homology detection. RAPTOR-np is a variant of RAPTOR without considering pairwise interactions when doing optimal alignment, but the pairwise score is still calculated based on the non-pairwise alignment. The corresponding weight factors and SVM model are optimized separately using the same sets of training samples. Compared with RAPTOR-np, RAPTOR is better in fold level and superfamily level and same in family level. Thus, we may conclude that a strict treatment of the pairwise interactions is necessary for fold level recognition or even superfamily level.

## 6 Computing Efficiency Issues

An outstanding advantage of our algorithm is that the memory requirement is just about $O(M^2n^2)$ and, at most of time, the computing time does not increase exponentially with respect to the sequence size. Figure 1 shows the CPU time of threading 100 sequences (chosen randomly from Lindahl's benchmark) with size ranging from 25 to 572 to a typical template 1191 of length 162 (with topological complexity[7] 3 and 12 cores). According to Xu *et al.*'s paper[7], the computing time of PROSPECT is $O(Mn^5)$ and its memory usage is $O(Mn^4)$. The observed memory usage of RAPTOR is $100 \sim 200M$ for most of threading pairs. Figure 1 shows that the computing

time of our algorithm increases very slowly with respect to the sequence size. In fact, we found out that our relaxed linear programming gave the integral solutions most of time or generated only a few branch nodes when the solution was not integral.

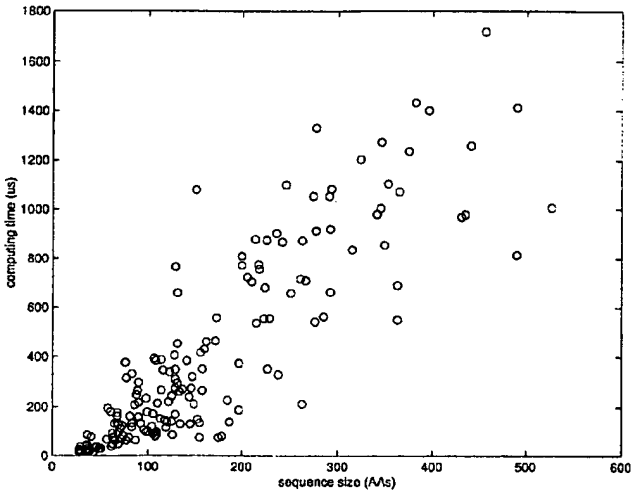


Figure 1: computing time of threading 100 sequences to template 1191.

# 7 Improving LP Efficiency

Protein threading is a very effective method to do fold recognition. Profile-based method runs very fast but could only recognize easy targets (Homology Modelling targets). Pairwise contact potential plays an important role in recognizing hard targets (Fold Recognition targets). However, if pairwise contacts are considered and variable gaps are allowed, then protein threading problem is NP-hard. Therefore, an efficient and exact algorithm is very indispensable for protein threading in order to do fold-level recognition. PROSPECT makes use of a clever divide-and-conquer algorithm to search for the globally optimal alignment between templates and sequences. Its computational complexity depends on the topological complexity of the template contact graph. It is very efficient for those templates whose contact graphs have low topological complexity. PROSPECT could effectively

exploit the sparseness of contact graphs. If the distance cutoff is 7A, there are about 3 quarters of contact graphes which has low complexity ($\leq 4$). However, PROSPECT couldn't deal with those templates with high topological complexity. The integer programming method could be used to deal with those templates with high complexity by exploiting the relationship between various kinds of scores contained in the energy function. However, for a long sequence and a complex contact graph, there are often tens of millions of variables involved in the integer program. Even for a topologically complex contact graph, there is often some regular part contained in this graph, i.e., this graph might have a subgraph which is topologically simple. Integer programming approach could not automatically take advantage of the partial regularity of the contact graph. We could use divide-and-conquer algorithm to align this kind of regular subgraph to the sequence first. In this paper, we would like to combine the integer programming approach and the divide-and-conquer algorithm. Before employing the powerful integer programming approach to formulate the problem, we first compress the contact graph to generate a dense contact graph by some graph reduction operation. During the process of reduction, simple dynamic programming and divide-and-conquer algorithm could be used to align one segment to the sequence.

## 7.1 HyperGraph-Based Integer Program Formulation

In this section, we will present a hypergraph-based integer program formulation which could be regarded as the generalization of the integer program formulation mentioned in the above sections. This kind of formulation could be used in two aspects. First, if the threading energy function takes the multiple-wise rather than pairwise contact potential into account, then the template contact graph becomes the hypergraph. We need to deal with a hypergraph based threading problem directly. Second, even if only the pairwise contact potential is considered, after graph reduction operation which would be discussed in details in the following sections, the reduced contact graph could become a hypergraph. In order to take advantage of the integer programming approach, we need to generate our simple graph based integer program formulation to the hypergraph-based formulation.

**Definition 7.1** *We use an undirected contact hypergraph* $HCMG = (V, HE)$ *to model the contact map graph of a protein template structure* , $V = \{c_1, c_2, ..., c_M\}$, *where* $c_i$ *denotes the* $i^{th}$ *core and* $HE = \{e = (c_{i_1}, c_{i_2}, ..., c_{i_k}) |$ *there is one multiple-wise interaction among* $c_{i_1}, c_{i_2}, ..., c_{i_k}\}$.

The only difference between the hypergraph-based threading problem and the simple graph based threading problem is that the energy function of hypergraph-based threading consists of the multiple-wise interaction preferences. If there is a multiple-wise interaction among $t_1, t_2, ..., t_k$, Let $P(s_1, s_2, ..., s_k, t_1, t_2, ..., t_k)$ denote the multiple-wise interaction score when aligning $s_i$ to template position $t_i$, then the objective of hypergraph-based threading is to minimize $\sum F(s_i, t_i) + \sum P(s_1, s_2, ..., s_k, t_1, t_2, ..., t_k)$.

For any $e = (c_{i_1}, c_{i_2}, ..., c_{i_k}) \in HE$, let $y_{e,l_1,l_2,...,l_k}$ indicate the multiple-wise interaction among $x$ variable $x_{i_1,l_1}, x_{i_2,l_2}, ..., x_{i_k,l_k}$. $y_{e,l_1,l_2,...,l_k} = 1$ if and only if core $c_{i_r}$ is aligned to sequence position $l_r$ $(r = 1, 2, ..., k)$. Then we could formulate the objective function of the hypergraph based threading as follows.

$$Min \sum x_{i_r,l_r} F(l_r, i_r) + \sum y_{(e,l_1,l_2,...,l_k)} P_{l_1,l_2,...,l_k,e}$$

Where $F(l_r, i_r)$ denote the single-wise score when core $i_r$ is aligned to sequence position $l_r$ and $P(l_1, l_2, ..., l_k, e)$ the multiple-wise interaction score when $e = (c_{i_1}, c_{i_2}, ..., c_{i_k})$ and core $c_{i_r}$ is aligned to sequence position $l_r$.

Let $D[i]$ denote the set of sequence positions that core $c_i$ could be aligned to and $R[i, j, l]$ the set of sequence positions that core $c_j$ could be aligned to given that core $c_i$ is aligned to sequence position $l$.

The constraint set is as follows:

$$\sum_{l \in D[i]} x_{i,l} = 1 \tag{22}$$

For any $e = (c_{i_1}, c_{i_2}, ..., c_{i_k}) \in HE$, $\forall l_r \in D[i_r]$, we have

$$x_{i_r,l_r} = \sum_{l_p \in R[i_r,i_p,l_r]} y_{(e,l_1,l_2,...,l_k)} \tag{23}$$

$$y_{e,l_1,l_2,...,l_k} \in \{0, 1\} \tag{24}$$

$$x_{i,l_i} \in \{0, 1\} \tag{25}$$

## 7.2 Contact Graph Reduction

For a long protein template and a long query sequence, the number of the integer variables is often very huge, which will take the integer program a very long time to find the optimal solution from the search space of exponential size. Before applying the integer programming approach to the threading problem, we try to decrease the number of the integer variables

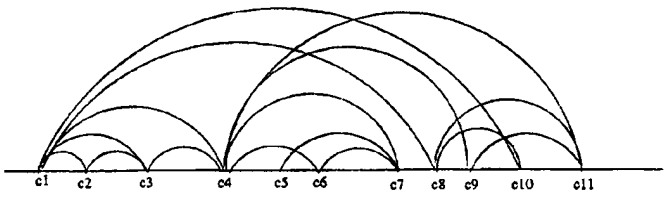


Figure 2: Template Contact Graph.

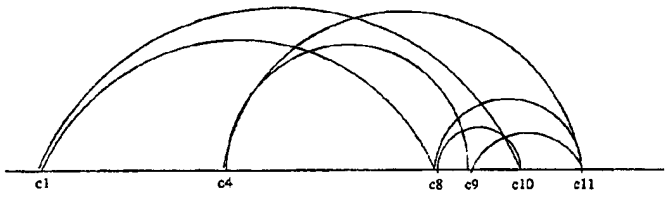


Figure 3: Compressed Contact Graph.

by employing graph reduction technique onto the template contact graph. This technique is effective because many template contact graphs are sparse or contain sparse subgraphs.

### 7.2.1 Graph Reduction Operation

If there are two cores $c_i, c_k (i < k)$ such that $\forall j (i < j < k)$, $N(c_j) \in \{c_i, c_{i+1}, ..., c_k\}$ and the subgraph formed by $c_{i+1}, c_{i+2}, ..., c_{k-1}$ is a graph with low topological complexity such as a nested graph, then we can first align segment ($c_i$,$c_k$) to the sequence by some algorithm with low computational complexity such as divide-and-conquer or dynamic programming and then treat the whole segment ($c_i$, $c_k$) as one edge when formulating the integer program. As shown in the original figure 2 and the resultant figure 3, segment ($c_1$,$c_4$) and segment ($c_4$,$c_8$) are reduced to two edges respectively cause the subgraphs formed by $c_1, c_2, c_3, c_4$ and $c_4, c_5, c_6, c_7, c_8$ are topologically simple. Segment ($c_1$,$c_4$) could be aligned to the sequence by simple dynamic programming algorithm. Segment ($c_4$,$c_8$) could be aligned to the sequence by divide-and-conquer algorithm within low degree polynomial time. This kind of graph reduction operation will be formulated as follows.

**Definition 7.2** *Given a template contact graph $CMG = (V, E)$, $|V| \geq 2$, a subset $RV = \{c_{i_1}, c_{i_2}, ..., c_{i_k}\}$ $(i_1 < i_2, ..., < i_k, k \geq 2)$ of $V$ is called* Reduced Vertex Set *if $\forall e = (c_l, c_p) \in E$ $(l < p)$, either at least one of $c_l$,$c_p$ is in $RV$ or there exists one $j$ such that $i_j < l < p < i_{j+1}$.*

**Definition 7.3** *Given a template contact graph $CMG = (V(CMG), E(CMG))$, and its* Reduced Vertex Set *$RV$, its* Reduced Contact Graph *$RCMG$ could be constructed as follows:*
*(i) $V(RCMG) = RV$;*
*(ii)$\forall v_1, v_2 \in V(RCMG)$, if $(v_1, v_2) \in E(CMG)$, then add $(v_1, v_2)$ to $E(RCMG)$;*
*(iii) For any segment $(c_{i_r}, c_{i_{r+1}})$, let $c_{i_{j_1}}$,$c_{i_{j_2}}$,...,$c_{i_{j_p}}$ denote all cores in $RV$ which are adjacent to at least one core within segment $(c_{i_r}, c_{i_{r+1}})$, we add one hyperedge $(c_{i_r}, c_{i_{r+1}}, c_{i_{j_1}}, c_{i_{j_2}}, ..., c_{i_{j_p}})$ into $E(RCMG)$.*

According to Definition 7.2 and Definition 7.3, we can see that for any given template contact graph, there are often many possible reduced contact graphes(see figure). Therefore, which one is the best in terms of computational complexity? Let $RV = \{c_{i_1}, c_{i_2}, ..., c_{i_p}\}$ denote the reduced vertex set of the contact graph after graph reduction operation. Let $T_{seg}$ denote the computational complexity of all segments $(c_{i_r}, c_{i_{r+1}})$,$(r = 1, 2, .., p$, when $r = p$, let $r + 1 = 1)$ and $T_{reduced}$ denote the computational complexity of the integer programming approach on the reduced contact graph. Ideally, the best graph reduction operation should minimize the maximum of $T_{seg}$ and $T_{reduced}$. It's easy to theoretically analyze how much time it will take for aligning all segments (see the next subsection). However, it's hard to estimate the computational complexity of the integer programming algorithm on the reduced graph though there is a trivial prediction that $T_{reduced} = O(n^p)$ ($n$ is the sequence size.). In practice, $T_{reduced}$ is far smaller than $O(n^p)$. There are two factors which are related to the computational time of the integer program. One is the size of the reduced vertex set, and the other is the complexity of the hyperedge, i.e., the number of cores contained in one hyperedge. Real protein data shows that a good reduction is to make $T_{seg} = O(Mn^k)$, $k = 3, 4, 5$ ($M$ is the number of cores) and limit the complexity of the hyperedge to at most 3. For $k = 3$ or the hyperedge complexity less than 3, the reduced contact graph is still a simple contact graph.

# 8 Summary

What is claimed is:

1. A method of protein threading by linear programming.

2. Three formulations of optimizing energy functions by linear programming.

3. A method for improving the linear programming formulation of the protein threading problem by graph reduction.

4. A method of protein threading by linear programming such that the solutions are more likely to be integral.

5. The method of any one of claims 1 - 4, further comprising the step of calculating a threading energy function.

6. The method of claim 5 further comprising the step of exploiting the relationship between various kinds of scores contained in the energy function.

7. The method of any one of claims 1 - 4, further comprising the step of performing a support vector machine (SVM) algorithm.

8. The method of claim 2, comprising the prior step of generating a dense contact graph.

9. A method of alignment comprising the steps of: formulating the protein threading problem as a large scale integer programming problem; relaxing this problem to a linear programming problem; and solving the integer program by a branch-and-bound method.

10. An apparatus for executing the method of any one of claims 1 - 9.

11. A system executing the method of any one of claims 1 - 9.

12. A computer readable memory medium storing software code executable to perform the method of any one of claims 1 - 9."

## References

[1] D.T. Jones. *J. Mol. Biol.*, 287:797–815, 1999.

[2] R.H. Lathrop. *Protein Engineering*, 7:1059–1068, 1994.

[3] D.T. Jones, W.R. Taylor, and J.M. Thornton. *Nature*, 358:86–98, 1992.

[4] A.Godzik, A.Kolinski, and J.Skolnick. *J. Mol. Biol.*, 227:227–238, 1992.

[5] S. Bryant. *Proteins: Struct. Funct. Genet.*, 26:172–185, 1996.

[6] T. Akutsu and S. Miyano. *Theoretical Computer Science*, 210:261–275, 1999.

[7] Y. Xu *et al. Journal of Computational Biology*, 5(3):597–614, 1998.

[8] D. Fischer *et al.* pages 300–318. PSB96, 1996.

[9] D. Kim, D. Xu, J. Guo, K. Ellrott, and Y. Xu. 2002. Manuscript.

[10] S.F. Altschul *et al. Nucleic Acids Research*, 25:3389–3402, 1997.

[11] R.M. Schwartz and M.O. Dayhoff. pages 353–358. Natl. Biomed. Res. Found., 1978.

[12] L. Holm and C. Sander. *Science*, 273:595–602, 1996.

[13] G.H. Gonnet *et al. Science*, 256:1443–1445, 1992.

[14] D.T. Jones. *J. Mol. Biol.*, 292:195–202, 1999.

[15] Laurenece A. Wolsey. John Wiley and Sons, Inc., 1998.

[16] L. Holm and C. Sander. *ISMB*, 5:140–146, 1997.

[17] N.N. Alexandrov. *Protein Engineering*, 9:727–732, 1996.

[18] M.S. Eldred *et al.* Technical Report SAND2001-3796, Sandia, 2002.

[19] S.H. Bryant and S.F. Altschul. *Curr. Opin. Struct. Biol.*, 5:236–244, 1995.

[20] T. Joachims. MIT Press, 1999.

[21] V.N. Vapnik. Springer, 1995.

[22] A.G. Muzrin *et al. J. Mol. Biol.*, 247:536–540, 1995.

[23] E. Lindahl and A. Elofsson. *J. Mol. Biol.*, 295:613–625, 2000.

[24] J. Shi, L. B. Tom, and M. Kenji. *J. Mol. Biol.*, 310:243–257, 2001.